Creation of a Computerized Interface to Control a Deposition System Subset Using LabVIEW and FieldPoint

Eduardo Moutinho
Energy Research Undergraduate Laboratory Fellowship Program
Colorado School of Mines
National Renewable Energy Laboratory
Golden, CO, 80401

August 15, 2002

Prepared in partial fulfillment of the requirements of the ERULF Program at the National Renewable Energy Laboratory.

Participant:

Signature

Research Advisor:

Signature

Research Advisor:

Signature

Introduction

In the field of renewable energy, computers control many different tasks and processes. Methods and processes are being updated and improved continually as new technologies emerge. LabVIEW is one of these technologies. Basically, LabVIEW is a graphically based programming language developed by National Instruments that is primarily used with process equipment. Because of its graphical backbone, LabVIEW is easier to use than the most text-based programming languages (Figure 1). This allows for programs to be developed quickly. Also, concepts such as debugging, troubleshooting, and layouts are easier to execute with LabVIEW's interface.

The National Renewable Energy Laboratory (NREL) is one of the several laboratories around the nation that are using LabVIEW to facilitate the use of process equipment. Specifically, Dr. David Albin, Dr. Ramesh Dhere, and I are using LabVIEW to automate a subset of the 3M process control system. This specific subset involves the pump-down procedure, in which specific valves are opened in sequence to bring the equipment to the desired pressure (Figure 2). This procedure also involves activating cooling water distribution and power to the system.

The program will be tested with a protoboard containing light-emitting diode (LED) lights, switches, and voltage sources to emulate the valves and switches of the 3M deposition system. The communication from the PC to the protoboard will be done via FieldPoint modules. FieldPoint is a system developed by National Instruments to allow LabVIEW to communicate with outside sources (Figure 3). Different FieldPoint modules work with different devices. We will be using modules designed for use with thermocouples and analog/digital sources.

By creating such a program, we will lay the foundation for the creation of a completely computerized interface for the 3M deposition equipment. A LabVIEW-based interface will permit for easy automation of the equipment. The basic deposition process can be done with a simple series of mouse-clicks. This eliminates the need for the scientist using the machine to manually conduct the sequences using its actual switchboard. This switchboard will be replaced with a virtual one on the PC that can still be manually controlled. But the machine will rarely be operated by manual control, since the automated procedure will be available. This saves the scientist time and effort, and greatly reduces the possibility of human error.

Materials and Methods

Tools Used to Develop the Software

The majority of this project was conducted on a Pentium-class PC with LabVIEW 6.1 installed. To familiarize myself with LabVIEW, I studied various manuals and tutorials. For the most part, I focused on the LabVIEW Basics series of tutorials written by National Instruments. These tutorials guided me on how to implement concepts such as loops and case structures. These concepts encompass the bulk of the final program that my project is based upon.

In addition to the office PC, I used another Pentium-class PC inside the laboratory. This laboratory PC acted as a test-bed for the final program (Figure 4). A series of FieldPoint modules mounted on din rails were connected to the PC via an Ethernet crossover cable. FieldPoint acts as a communication gateway between the PC and whatever the LabVIEW software is supposed to control. Each FieldPoint module has a specific model number and function. Our setup consisted of terminal bases for holding the modules in place, a unit to communicate with the computer, a thermocouple unit, a digital input unit, an analog input unit, and a power supply unit.

The modules were connected to a simple protoboard wired to a series of LED lights, toggle switches, and voltage dividers. The protoboard was used as a means of emulating the actual 3M system subset. LED lights represented valves, toggle switches represented switch closures, and the voltage dividers represented manometers. By using the protoboard as an effective means of emulating the subset, we were able to effectively test the program without having to hook it up to the actual system. Later in the project development phase, the protoboard was replaced with a small box to house all of the LED lights, voltage dividers, and switches. This was done for both aesthetic and practical reasons. Therefore, the project was developed and completed using both the office and laboratory computers. The office computer acted as the development base for the actual software, while the laboratory computer acted as the test unit.

Planning the Software

Before actual program development began, I met with my mentors to discuss what the software should do in general. During the meeting we diagrammed the basic layout of the 3M system subset, and we talked about the tasks that the program was required to do. By doing this, we were able to derive a statement of the project's scope, and specify individual tasks. Next, we began to discuss what kind of methods I would use to develop the actual program. We thought of using while (continuous) loops and Boolean (true or false) ideology. Sequence structures were also brought up as a possibility. We also created a possible layout, which helped me to visualize what the code for the program might look like.

After this meeting, I met with Dr. Dhere to observe a simulation of the subset using the 3M process equipment. Dr. Dhere showed me the actual switchboard on the equipment and how it was used to pump the system down.

LabVIEW

The LabVIEW programming language consists of two main parts: the front panel and the block diagram (Figure 1). The front panel represents the aesthetic viewpoint of the programming, since it is used to build and design the interface of the program. The front panel portion of the programming is the simplest part. Basically, to build an interface, the programmer right clicks on any empty portion of the panel, selects what is wanted from a menu, and drags the icons onto a screen. A wide variety of options are available, such as thermometers, graph generators, switches, and indicators. After the interface for the program has been completed, the block diagram portion of the programming goes into effect.

The block diagram is where the programming aspect of LabVIEW is implemented. Without the block diagram, the program will not function. Instead of writing various lines of code to connect objects on the program interface together, the programmer uses virtual “wires.” Icons with inputs represent the objects on the diagram. These inputs are then “wired” to other objects. As in text-based programming, loops, case structures, Boolean ideology, and arrays are available. The programmer creates loops and case structures by making boxes around the icons that are to be placed within them. This method of programming allows software to be developed quickly and modified easily. Creating a while loop within a case structure within a while loop in LabVIEW takes only seconds, in comparison to the more tedious and time-consuming method of writing it out in a text-based language.

Software Development

In the beginning stages of development, I focused on learning how to make the laboratory PC communicate with the LED lights connected to the protoboard. Although this was a simple goal, it proved to be a somewhat difficult task. In addition to developing the actual code to make a light turn on and off, which was relatively simple, the FieldPoint modules had to be detected and configured by the computer. This was done using a program called FieldPoint Explorer, also developed by National Instruments. With Explorer, I was able to detect the FieldPoint modules and their inputs. Also, I was able to change the names of each input to reflect what they were connected to. So I renamed certain inputs to "LED 1," "LED 2," and so forth. This allowed tags to be created more easily within LabVIEW, since the devices and inputs had to be tagged within the code for the program to know the virtual addresses of the hardware, allowing for communication to occur between both mediums.

The initial LED test worked, yet there was a significant delay between the computer and the protoboard. When the switch within the program was activated, the light turned on two seconds later. This was perplexing, since the code was extremely simple. To remedy the problem, my mentors contacted Paul Sweat of National Instruments to help with the situation. He not only alleviated the issue with the delay, he also provided guidance in developing the final program.

When the LED test was completed and all the FieldPoint modules were detected and operational, I proceeded to work on the final program. I decided to use state machines to formulate it. A state machine is a programming concept that involves using case structures within while loops to create an event-based program. This means that the user can define what the program does by pushing a button, or a series of buttons. There is little to no dependence on forced sequences over which the user has little control (National Instruments LabVIEW Development Library). The National Instruments LabVIEW Support Database states that “almost all programs that have more than a rudimentary user interface are state machines--they wait for user input, which can be any number of possible values, and then take appropriate actions based on this input” (National Instruments LabVIEW Development Library). The program’s manual interface would require the use of switches, which in essence are buttons. These switches would be activated by a user in whatever order they chose. Also, the program’s sequence mode would be a user-activated one, triggered by a button. Therefore, the state machine concept turned out to be a perfect method to use in the development of the program.

Results

Manual Switchboard

Using the aforementioned methods, we created a final program that works well. The manual switchboard worked flawlessly after some modification. The first version of the manual switchboard would lock up after an interlock failure, leaving whatever was activated still activated. This was an inaccurate depiction of what the program was required to do. However, after some simple modifications were made to the code, the problem was solved.

Sequence Mode

The sequence portion of the program proved to work well, after a lengthy set of modifications. The state machine concept proved to be a viable method of implementing a user-initiated sequence. The first version of the sequence mode did not function because of inconsistencies with the Boolean ideology. Case structures were not being activated at appropriate times, because the signals were not reaching their intended locations. After the problem with the Boolean inputs was fixed, the program continued to have problems functioning correctly. The actual case structure that housed the sequence was not proceeding from step to step. By using the debugging tool in LabVIEW, which basically slows the program down and uses many dots to show where each signal is going, I was able to see the problem. The targets connecting each case (step) were inconsistent, causing the program to get confused. The targets were fixed, however, new problems arose.

The lights representing the valves were illuminating in order, yet they were not accurately representing the sequence. This presented the biggest challenge since I knew how to fix the problem but not how to actually code it with LabVIEW. I wanted to use the same input in two different locations, and I was not sure how to do that. This forced me to do some searching on my own to find the answer.

Fortunately, I found the solution to my problem. I used local variables to represent inputs at more than one location. This allowed me to use the same input in different cases, which facilitated emulating the exact sequence I desired. Still, challenges continued to present themselves. The lights would illuminate immediately without waiting for the correct conditions to be met. This was quite a daunting predicament. This problem was solved by using while loops around the objects that represented the lights. By having a while loop at those locations, the program would reach that specific juncture and then wait for the correct conditions to be met. When all of these challenges were successfully dealt with, I was able to get the correct sequence to work within the program.

Nevertheless, a few additional problems remain. For example, the interlock check within the sequence is a lot more difficult to implement because of the event-driven nature of the code. Therefore, when the program is waiting for a condition to be met, it is constantly looping a small section of it. This section does not have an interlock check within it. Theoretically, if an interlock error were to occur, the program would not pick it up at that juncture.

In addition, the user-interface could be improved. At the moment, there is a toggle switch between manual and sequence mode. You cannot switch the toggle from manual to sequence or vice versa if you are running the program. To switch between the two modes, you have to shut down the program and switch while it is stagnant. The program works well, but this task is a bit tedious.

Discussion and Conclusion

The main goals for creating a program to emulate a subset of the 3M system were met. The program does exactly what it is supposed to in an efficient manner. Manual control is flawless, while the sequence mode shows that a process can be automated using LabVIEW. In a matter of ten weeks, I was able to learn the LabVIEW programming language, establish a connection between a PC and an outside source by using FieldPoint, and create a program to emulate a subset of the 3M process equipment. This shows that

LabVIEW is indeed a powerful and easy-to-learn tool that can be used effectively in the field of process control.

Throughout the program development period, I was required to make various modifications to the code. I was asked to add things to the program quickly and modify complex structures, such as the main sequence. With LabVIEW, I was able to do these things with incredible ease. LabVIEW's graphical environment makes it easy to make changes. Also, the graphical layout simplified the troubleshooting process, as I was able to see where each signal went within the program, by using the debugging mode.

With more time, it might be possible to fix some of the unsatisfactory features. These include the interlock problem with the sequence mode and the problem associated with switching between the manual and sequence modes. I have some ideas about how to fix these problems. For example, I believe the interlock problem can be fixed by developing a sub-VI (subroutine) to act as an interlock check. That way, the user can run multiple sub-VIs within the code making sure that the interlocks are being checked at every possible juncture. Nevertheless, these problems are relatively small. The program accurately simulates manual control, and the sequence works to exact specifications.

The software that I developed represents the first step in a long process to fully automate the 3M process system with LabVIEW. Although I have emulated only a small subset of the system, expanding the software to encompass the entire equipment will be relatively simple. This is so because the subset involved most of the devices that will have to be controlled. Developing the final software for the system will be similar to creating the software I worked on, only on a larger scale.

Acknowledgements

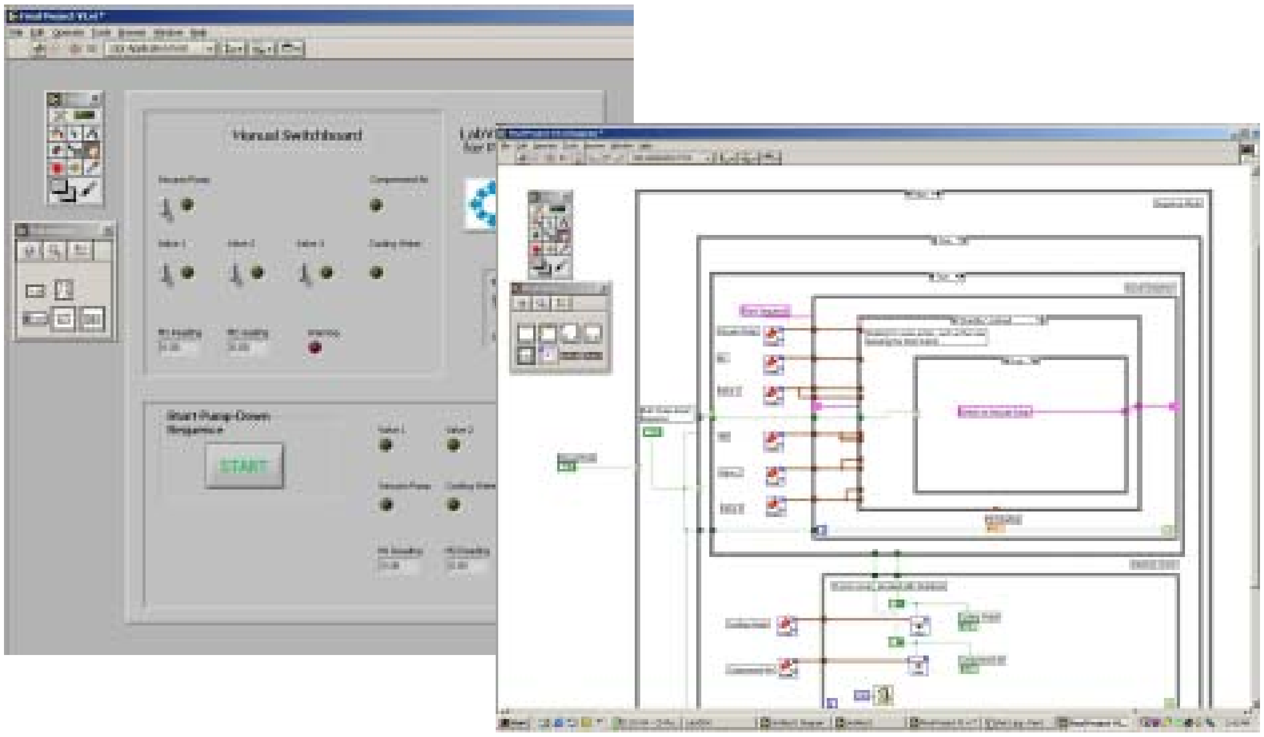
First and foremost, I would like to thank Dr. David Albin and Dr. Ramesh Dhere for being my mentors. They helped me tremendously throughout each phase of this project. Without them, I would not have been able to successfully conduct the project. Also, I'd like to acknowledge Paul Sweat, Matthew Page, and Marc Landry for helping me get a firm grasp on LabVIEW and FieldPoint. Next, I would like to thank NREL, the ERULF program, and the United States Department of Energy for providing me with this opportunity to work at a national laboratory with an excellent staff. Last and not least, I'd like to acknowledge the people who run the ERULF program at NREL. These people include Linda Lung, Patrisia Navarro, and Wilbur Sameshima for all their contributions. This summer has provided me with wonderful experiences that have helped me develop as a young adult, and I cannot give enough thanks for that.

References

National Instruments LabVIEW Development Library. Using a State Machine (Event Driven) Architecture. Retrieved Jul. 3, from National Instruments LabVIEW Website: http://zone.ni.com/devzone/taskdoc.nsf/2d17d611efb58b22862567a9006ffe76/8c4eecacf084f8e986256802007b9186?OpenDocument.

Figures

Example of LabVIEW Software Interface



Front Panel (Left) and Block Diagram (Right)

Figure 1

Layout of 3M System Subset

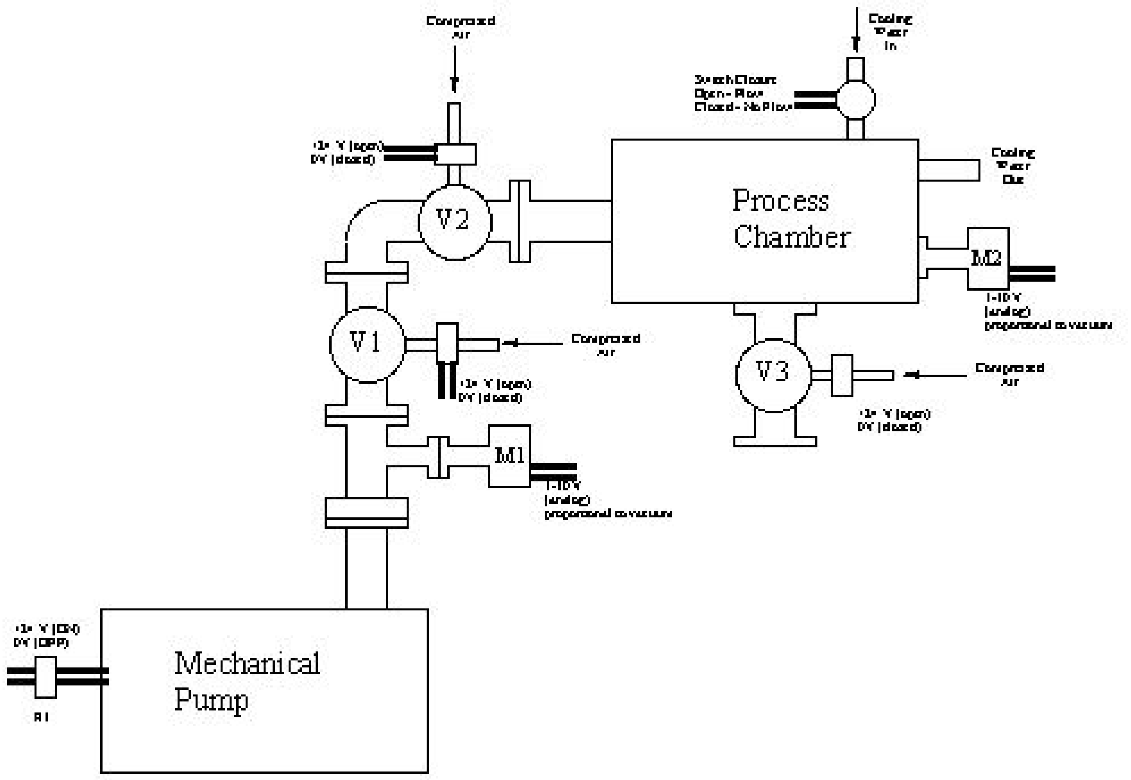


Figure 2

FieldPoint Modules in Test Bed Setup

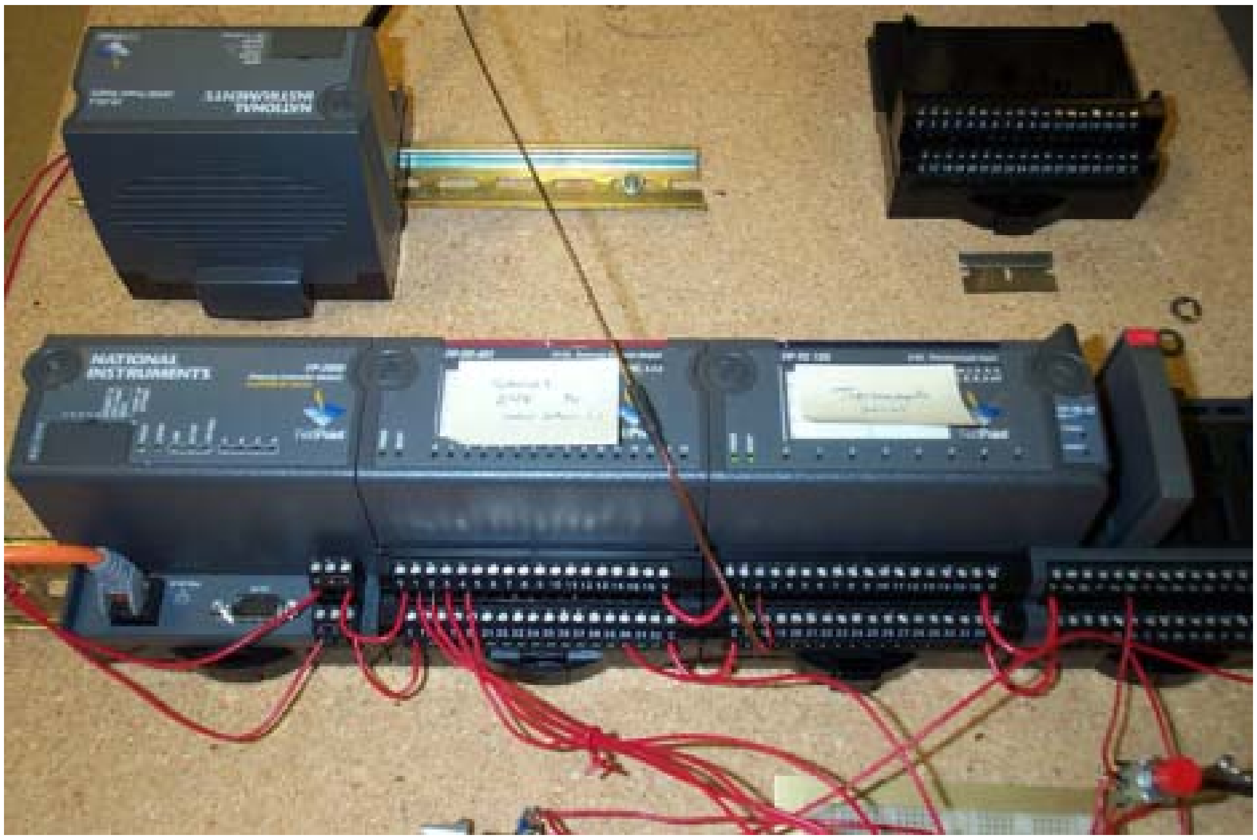


Figure 3

Test Bed with FieldPoint Modules, Protoboard, and PC

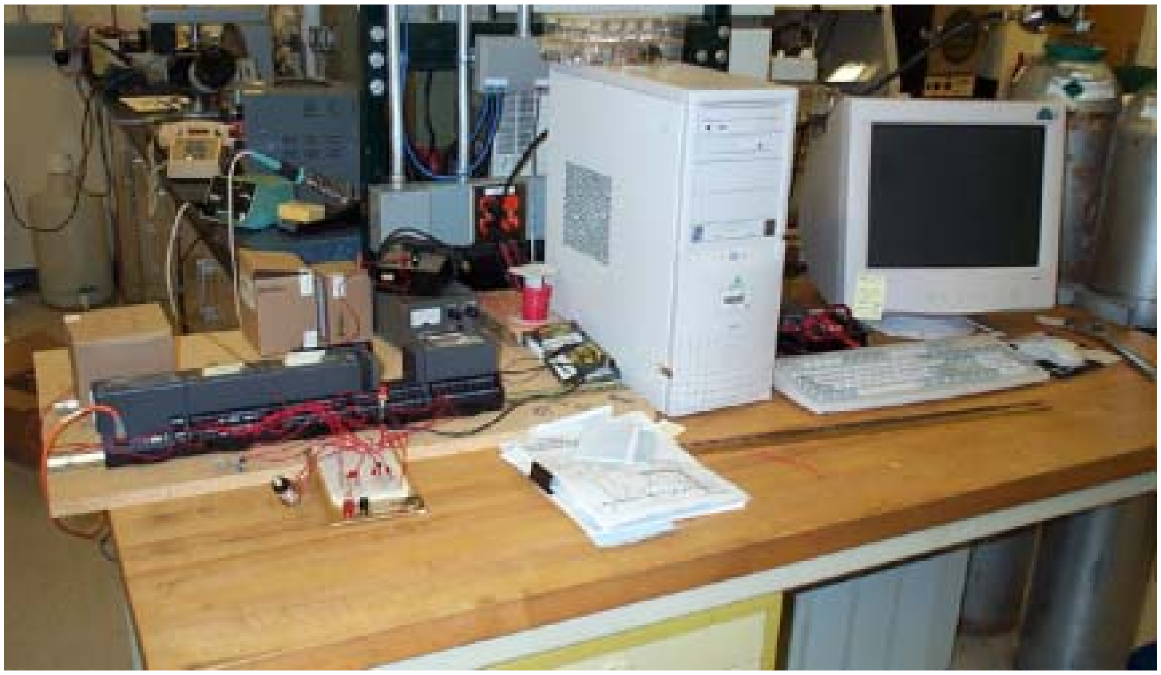


Figure 4